IET Journals
The Institution of Engineering and Technology

High-Throughput and Compact Reconfigurable Architectures for Recursive Filters

ISSN 1751-8644
doi: 0000000000
www.ietdl.org

*Vaishali Shinde, Ganesh Jai Kumar, Daniel Valencia, Amirhossein Alimohammad**

*Department of Electrical and Computer Engineering, San Diego State University, 5500 Campanile Drive, San Diego, U.S.A * E-mail: aalimohamad@sdsu.edu*

Abstract: This article presents various high-throughput reconfigurable architectures and their hardware implementation characteristics for infinite impulse response (IIR) filters. It is known that finite wordlength effects can be alleviated, to some extent, by realizing a high-order IIR filter using the cascade of lower-order filter sections. We utilize the cascade structure of the first-order and second-order filters and apply a set of optimization techniques, such as cutset retiming, lookahead transformations, and interleaving for high-throughput realizations of IIR filters. Since the cascade structure may require a relatively large number of computational resources and storage elements, which depends linearly on the number of sections, we also present a filter processor architecture for the compact implementation of IIR filters. Filter architectures are developed in the fully-parameterizable fixed-point representation and verified against their synthesizable Verilog descriptions. We present the implementation results of the transformed filter architectures on a Xilinx Virtex-7 field-programmable gate array (FPGA). To the best of our knowledge, this is the first presentation of various high-throughput and compact IIR filter architectures and their FPGA implementation characteristics.

1 Introduction

A digital filter is a linear time-invariant system that can be characterized by a constant-coefficient difference equation. The discrete time-domain difference equation can be written as:

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] - \sum_{k=1}^{N} a_k y[n-k], \qquad (1)$$

where $y[n]$ is the filter output at a discrete time instance n, $x[n-k]$ denotes the current and previous filter input samples, k indicates the number of time delays, b_k and a_k denote the k-th feed-forward and feedback filter coefficients, respectively, and $y[n-k]$ denotes the previous output samples of the filter. When at least one of the a_k coefficients is non-zero, the scaled output samples are fed back into the input to make a contribution to the next output values. This has the effect that an impulse applied to the filter results in a response that never decays to zero (though usually at diminished amplitudes) and hence, are called infinite impulse response (IIR) or recursive filters. The difference equation (1) states that the output depends on both the present and previous M inputs and the previous N outputs, where N is the order of the filter. Typically, the recursive coefficients a_k's are scaled to yield $a_0 = 1$.

The process of design and implementation of IIR filters can be divided into three steps: approximation, realization, and implementation. In the approximation step, the values of the filter's coefficients are approximated through a theoretical optimization procedure for a target filter specification. Design tools, such as Mathworks' Matlab's Filter Design Toolbox, provide an extensive library of functions to assist designers with this step. The filter design software approximates the filter coefficients with a high degree of precision, usually in double-precision floating-point format. However, for efficient digital hardware implementations, fixed-point representation of signals and arithmetic are commonly used. Unfortunately, fixed-point IIR filters are highly susceptible to a range of degrading finite wordlength effects. The goal of a fixed-point IIR-filter design is to quantify and minimize the finite wordlength effects, including arithmetic round-off errors and register overflow. The realization step addresses the selection of a filter structure that fulfills the design specifications under finite wordlength effects of filter coefficients and intermediate signals.

It is known that higher-order (larger than two) IIR filters are increasingly susceptible to coefficient quantization. One practical approach to mitigate the finite wordlength effects, to some extent, is to use cascade structures. That is, an IIR filter realized using a series of M lower-order IIR filters as opposed to one high-order section. More specifically, a high-order filter can be implemented using the cascade of second-order sections (SOSs) or "biquads" and a first-order section (FOS) for an odd-ordered filter. Each SOS and FOS may be realized using various structures, meaning that their difference equations can be computed differently. While higher-order filters can be realized in a parallel structure by computing the second-order and/or first-order sections concurrently, the cascade structure is generally preferred. The parallel realization of smaller order sections have greater probabilities of overflow due to the summation of all sections' outputs. It is also known that the cascade structure offers more control over the location of filter zeros and that for certain filter designs, the cascade structure may require fewer multiplications [1] [2]. The various SOS and FOS structures vary in terms of their sensitivity to finite wordlength effects, hardware resource requirements, their operating speeds, and their latencies and hence, each carrying relative advantages and disadvantages.

The implementation step involves the construction of the filter based on the chosen filter structure using digital hardware components, such as adders, multipliers, and registers (delay elements). The hardware implementation of filter structures using digital elements can be explained in terms of their architectures. For example, in the direct architecture implementation, an IIR filter can be implemented using a series of M sections. However, the direct implementation may not be particularly suitable for the compact realization of filters as the resource utilization is directly related to the order of the filter.

This article focuses on the efficient hardware implementation of various high-throughput and compact reconfigurable architectures for IIR filters. Recently, hardware implementations of digital filters using stochastic computing [3] [4] as well as more traditional filter implementations [5] [6] have received attention. We apply various transformations to the cascade IIR filter structure, including retiming, pipelining, lookahead transformations, and interleaving,

for their high-throughput implementations on a field-programmable gate array (FPGA). These architectural transformations are applicable to a wide variety of filter structures and architectures. For a compact implementation of the cascade structures, we propose a compact filter processor architecture utilizing folding transformation in which the operations of M sections of an IIR filter can be mapped onto and executed by $1 \leq K < M$ sections. Filter architectures are developed in the fully-parameterizable fixed-point representation and are verified against their synthesizable Verilog descriptions.

The rest of this article is organized as follows. Section 2 presents the retimed and pipelined architectures for direct-form structures. Sections 3 and 4 discuss the IIR filter architectures utilizing look-ahead transformation and C-slow transformation, respectively. Section 5 presents the implementation results of various high-throughput IIR filter architectures. Section 6 presents the architecture and implementation results of the proposed time-multiplexed reconfigurable IIR filter processor. Finally, Section 7 makes some concluding remarks.

2 Retimed Direct-Form Architectures

In a cascade structure, the most straightforward realization of a filter section is the direct-form I (DF-I) structure, in which the difference equation (1) is evaluated using two cascade sections without any intermediate transformations. Fig. 1(a) shows the DF-I SOS, where the memory elements are labeled with *D*, and *g_in* and *g_out* denote the input and output scalar coefficients, respectively. Retiming is the process of systematically relocating registers in the design to shorten the critical path delay of an architecture while the input-output relationship is preserved [7]. For the feed-forward section of an IIR filter, registers can be added across any feed-forward cutsets to achieve a desired level of pipelining without changing the transfer function (at the expense of additional latency). The bottleneck with recursive filters is that, fundamentally, delay elements cannot be inserted in the feedback section of the filter. This is because if a feedback sample is required at a time instance, then delaying this via some cutset registers will lead to a non-causal system. Therefore, one of the limitations of IIR filters is that they set a bound on the sample frequency at which an implementation of the filter can operate.

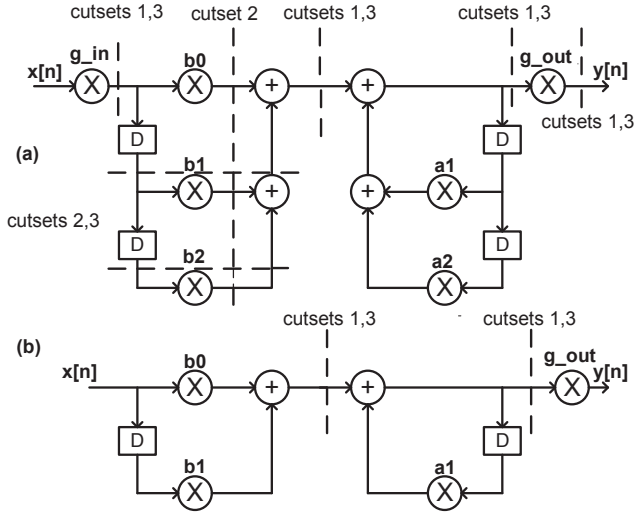

Fig. 1: Pipelined and retimed (a) SOS and (b) FOS structures in DF-I form.

In the DF-I SOS structure shown in Fig. 1(a), registers can be added at any of the shown cutsets. In our hardware implementations, the selective addition of cutset registers is controlled by the parameter *add_cutset*, which takes 1, 2, or 3 to realize a particular cutset in the direct-form structures. Due to the feedback structure, the minimum critical path delay of the retimed structure is $T_m + 2T_a$, where T_m and T_a denote the delay of a multiplier and an adder, respectively. The critical path can be seen as the combinational path beginning at the *a2* coefficient multiplier, the first coefficient product adder, and the accumulating adder that adds the feedback and feed-forward results together. The critical path delay would be the propagation delay through that combinational path, which specifies the minimum clock period. Because the exact propagation delay of each computational element depends on various factors, such as the operation topology and bit widths, specific values are not commonly used when referring to the critical path delay. Fig. 1(b) shows the pipelined DF-I FOS. Note that further addition of feed-forward cutsets to the DF-I FOS structure will not improve the speed of the cascade structure as it is limited by the critical path delay of the SOS.

A valuable property of the DF-I structure implemented in two's complement arithmetic is that there is only one summation point in the structure and since overflow naturally "wraps around" from the largest positive to the largest negative number and vice versa, then as long as the final result $y[n]$ lies in the dynamic range of the arithmetic, overflow is avoided, even when there are overflows in partial sums [8].

As IIR filters are linear systems, the order of the feed-forward and feedback sections can be interchanged, which results in the direct-form II (DF-II) structure. Fig. 2(a) shows the pipelined and retimed DF-II SOS structure with the critical path delay of $T_m + 2T_a$. Fig. 2(b) shows the pipelined DF-II FOS structure. Theoretically there is no difference between the DF-I and DF-II structures and the two structures are equivalent. Note that the DF-II structure, which is canonical with respect to delay (i.e., uses the minimum number of delay elements), may require just as many or more memory elements as the DF-I structure, even though the DF-I uses twice as many delay elements. This is because it is common for the feedback section to provide a large signal gain, especially for the filters with sharp transitions in their frequency response. Since in the DF-II structure the feedback section precedes the feed-forward section, it increases the possibility of arithmetic overflows and hence, the signal entering the delay line typically requires a larger dynamic range than the output signal $y[n]$. Although this gain is then compensated by attenuation in the feed-forward section, if the number of bits in the delay elements is increased (e.g., doubled), the benefit of halving the number of delay elements relative to the DF-I structure is negligible.

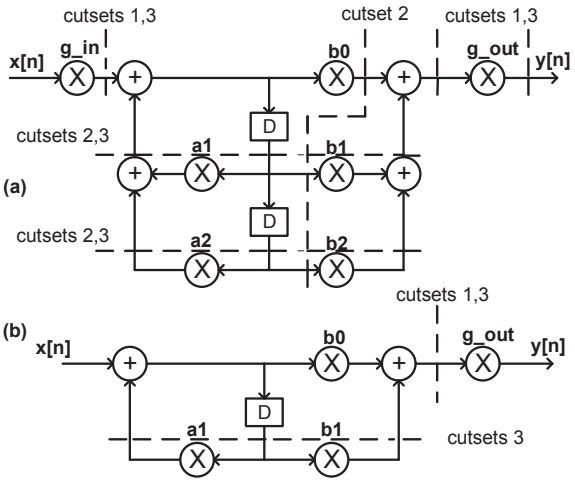

Fig. 2: Pipelined and retimed (a) SOS and (b) FOS structures in DF-II form.

Table 1 gives the characteristics and the implementation results of various FOS and SOS structures in both DF-I and DF-II forms for (*WI=5,WF=11*) fixed-point format on a Xilinx Virtex-7 XC7VX330T FPGA, where *WI* and *WF* denote the integer and the fractional bit-widths of the fixed-point representation of signals, respectively,

(A, M, D) denotes the number of adders, multipliers, and delay elements, L denotes the latency in clock cycles, and the maximum clock frequency is in MHz. The maximum clock frequency for each structure can be derived as $1/T_c$, where T_c is the critical path delay. We will use the same fixed-point format and FPGA device for the implementation of all other filter architectures.

Table 1 Characteristics and the implementation results of various FOS and SOS structures in DF-I and DF-II forms

Design	(A,M,D)	Slice Regs.	Slice LUTs	DSP48	L	Freq.
DF-I SOS	$(4, 6, 4)$	16	74	6	0	196
DF-II SOS	$(4, 6, 2)$	16	75	6	0	214
Ret. DF-I SOS	$(4, 6, 7)$	64	77	6	3	215
Ret. DF-II SOS	$(4, 6, 6)$	91	74	6	2	204
DF-I FOS	$(2, 3, 2)$	16	40	3	0	237
DF-II FOS	$(2, 3, 1)$	16	40	3	0	265
Pipe. DF-I FOS	$(2, 3, 3)$	32	40	3	1	263
Pipe. DF-II FOS	$(2, 3, 2)$	32	41	3	0	252

3 Look-ahead Transformed Architectures

Fundamentally, the feedback structures represent throughput bottlenecks and hence, a recursive filter may not be able to meet the target sampling requirements of the design. One technique for adding additional delay elements into the feedback section while conserving the filter's original transfer function is by utilizing the lookahead transformation (LAT) [9]. The LAT is an algebraic modification of the algorithm, which looks ahead and substitutes the expressions for previous output values in the IIR difference equation. The additional pipeline registers in the LAT architecture can then be retimed to reduce the original combinational path at the expense of increased computational complexity. For example, consider the SOS equation (1) for M = :

$$\begin{aligned} y[n] = b_0x[n] + b_1x[n-1] + b_2x[n-2] \\ - a_1y[n-1] - a_2y[n-2]. \end{aligned} \tag{2}$$

The terms for $y[n-1]$ and $y[n-2]$ can be written as:

$$\begin{aligned} y[n-1] = b_0x[n-1] + b_1x[n-2] + b_2x[n-3] \\ - a_1y[n-2] - a_2y[n-3], \end{aligned} \tag{3}$$

$$\begin{aligned} y[n-2] = b_0x[n-2] + b_1x[n-3] + b_2x[n-4] \\ - a_1y[n-3] - a_2y[n-4]. \end{aligned} \tag{4}$$

By substituting $y[n-1]$ and $y[n-2]$ into the SOS equation (2), the LAT can be written as:

$$\begin{aligned} y[n] &= b_0x[n] + (b_1 - a_1b_0)x[n-1] \\ &+ (b_2 - a_1b_1 - a_2b_0 + {a_1}^2b_0)x[n-2] \\ &+ ({a_1}^2b_1 - a_1b_2 - a_2b_1)x[n-3] \\ &+ ({a_1}^2b_2 - a_2b_2)x[n-4] \\ &+ (2a_1a_2 - {a_1}^3)y[n-3] \\ &+ ({a_2}^2 + {a_1}^2a_2)y[n-4]. \end{aligned} \tag{5}$$

Figs. 3 and 4 show the lookahead transformed SOS and FOS structures and their retimed structures in DF-I and DF-II forms. The critical path delays for the original LAT SOS and FOS structures in DF-I form are $(2T_m + 5T_a)$ and $(2T_m + 3T_a)$, respectively. The retimed LAT SOS and FOS structures in DF-I form both have a lower critical path delay of $\max(T_m, 2T_a)$. Similarly, the original LAT SOS and FOS structures in DF-II form have critical path delays of $(3T_m + 3T_a)$ and $(3T_m + 2T_a)$, respectively. Finally, the retimed LAT SOS and FOS structures in DF-II form show a decreased critical path delay of T_m. Table 2 gives the characteristics and the implementation results of the retimed LAT structures for the SOS structures in DF-I and DF-II forms.

Table 2 Characteristics and the implementation results of the LAT SOS structures in DF-I and DF-II forms

Design	(A,M,D)	Slice Regs.	Slice LUTs	DSP48	L	Freq.
LAT DF-I SOS	$(6, 8, 15)$	192	116	8	3	322
LAT DF-II SOS	$(6, 8, 16)$	208	118	8	4	315

Scattered look-ahead transformation (SLAT) [10] was proposed as a more numerically-stable realization of the lookahead transformation and its revised clustered look-ahead transformation [11]. The details of the SLAT can be found in [12]. Figs. 5 and 6 show the SLAT of the SOS structures and their retimed structures in DF-I and DF-II forms, respectively. By introducing more delay elements to the structure, as a result of the LAT, the node transfer theorem can be utilized to retime the delay elements in the design. According to the node transfer theorem [13], in a linear time-invariant system, a delay element can be moved from the input of a node to each of its outputs. Thus, after retiming, the critical path delays of the DF-I and DF-II structures are reduced to $\max(T_m, 2T_a)$ and T_m, respectively. Table 3 gives the characteristics and the implementation results of the retimed SLAT for the SOS structures in DF-I and DF-II forms.

Table 3 Characteristics and the implementation results of the SLAT SOS in DF-I and DF-II structures

Design	(A,M,D)	Slice Regs.	Slice LUTs	DSP48	L	Freq.
SLAT DF-I SOS	$(8, 10, 21)$	224	152	10	3	323
SLAT DF-II SOS	$(8, 10, 22)$	256	152	10	4	313

4 Interleaved IIR Filter Architectures using C-Slow Transformation

Interleaved architectures can be efficiently utilized in instances that the circuit clock is at least $C \geq 2$ times faster than the sampling clock. To interleave and process C independent streams of input data using the same cascade structure, C-slow transformation can be effectively utilized, in which every register in the structure is replaced with $C \geq 2$ registers [9]. This replacement does not change the input-output relationship, however, it reduces sampling frequency by a factor of C as the input is sampled at every C clock cycles. Hence, the overall throughput remains unaffected. Retiming can then be applied after C-slow transformation of the structure by relocating the registers to reduce the critical path delay at the expense of additional registers and increased latency. To apply C-slow transformation to the filter structures in DF-I and DF-II forms, every register is replaced with $C = 3$ and $C = 4$ registers, respectively, to achieve the critical path delay of $\max(T_m,T_a)$. Note that the greater values of C will not improve the performance. Fig. 7 shows the retimed 3-slow SOS and FOS structures in DF-I form and the retimed 4-slow SOS and FOS structures in DF-II form. Table 4 gives the characteristics and the implementation results of the retimed 3-slow and 4-slow structures in DF-I and DF-II forms.

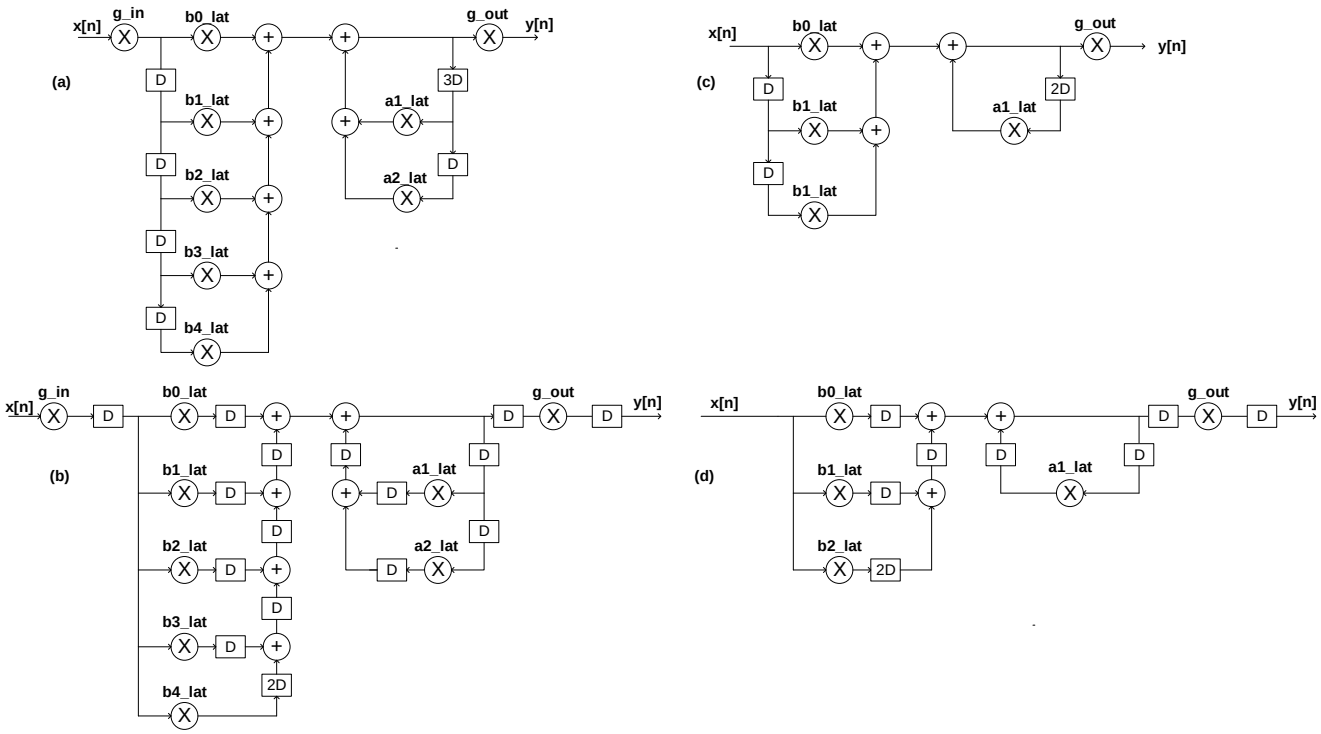

Fig. 3: (a) Look-ahead transformed SOS structure in DF-I form, (b) the retimed SOS structure in DF-I form, (c) the look-ahead transformed FOS structure in DF-I form, and (d) the retimed FOS structure in DF-I form.

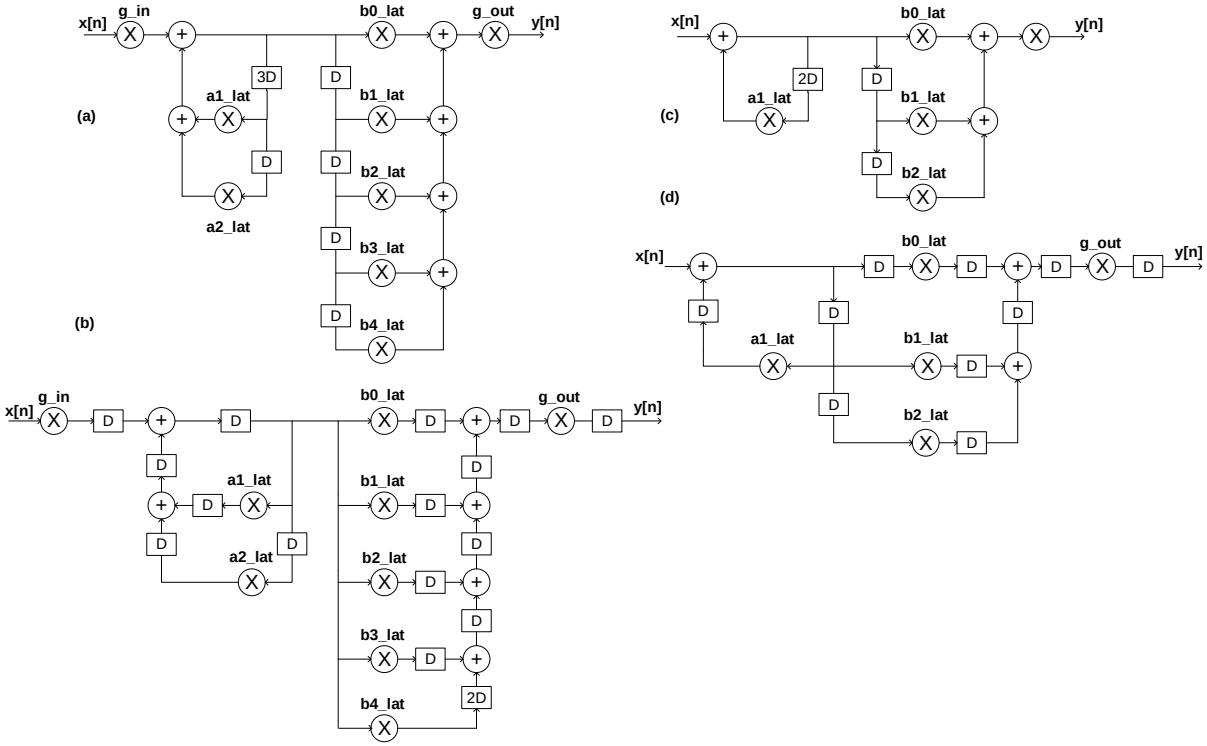

Fig. 4: (a) Look-ahead transformed SOS structure in DF-II form, (b) the retimed SOS structure in DF-II form, (c) the look-ahead transformed FOS structure in DF-II form, and (d) the retimed FOS structure in DF-II form.

5 High-Throughput IIR Filter Implementations using Various Filter Structures

Fig. 8 shows the pipelined cascade filter structure using M SOSs in which a register is added at the output of each section. To reduce the probability of overflow, the input signal can be scaled with a scaling factor before it is passed to the first SOS and the output from each section is optionally scaled before it is passed to the next section. In Fig. 8, *g_in* denotes the input scaling factor, *g_M* denotes the output scaling factor, and *Coeff_M* denotes the set of filter coefficients for the M-th section. The scaling factors and filter coefficients are determined during the approximation step. Each section (SOS or

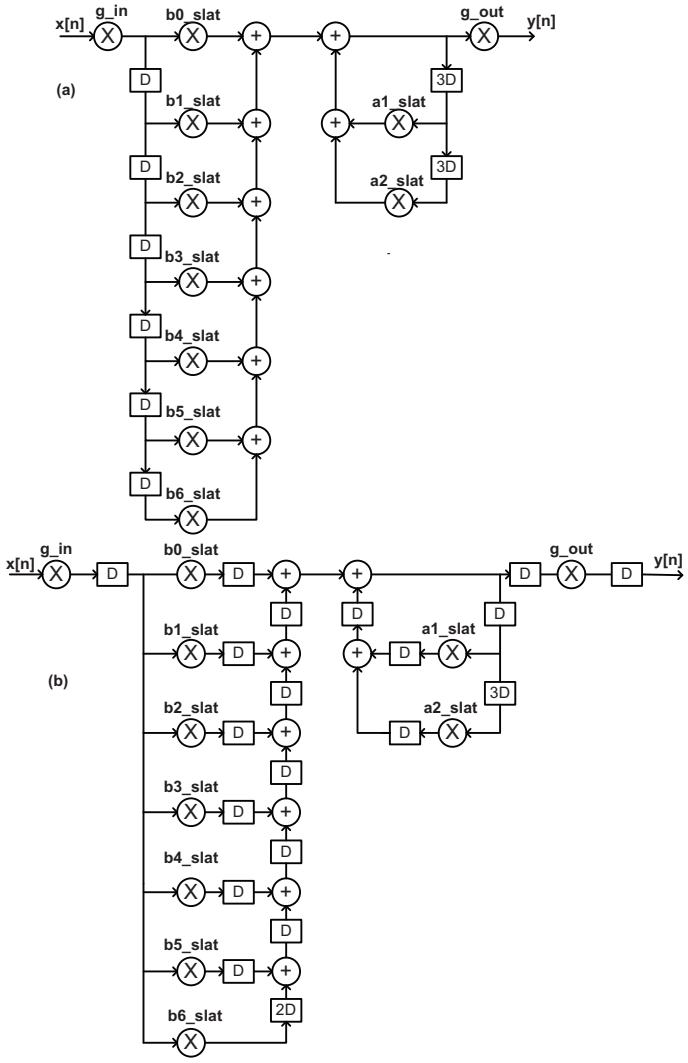

Fig. 5: (a) Scattered look-ahead transformed DF-I SOS and (b) its retimed structure.

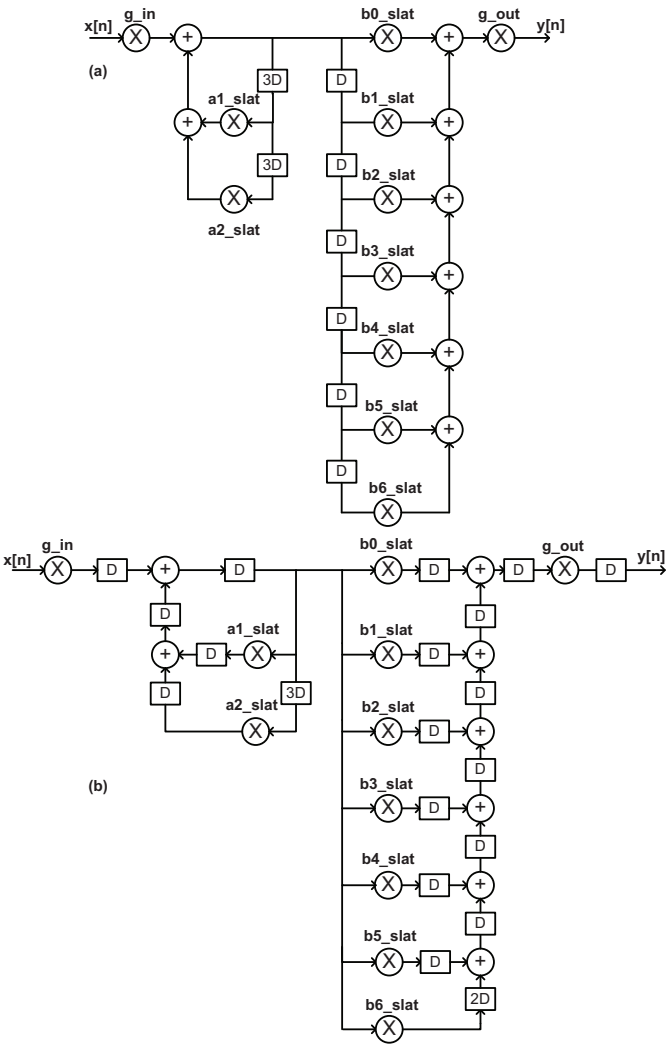

Fig. 6: (a) Scattered look-ahead transformed DF-II SOS and (b) its retimed structures.

Table 4 Characteristics and the implementation results of the 3-Slow and 4-Slow structures in DF-I and DF-II forms, respectively

Design	(A,M,D)	Slice Regs.	Slice LUTs	DSP48	L	Freq.
3-SLOW DF-I SOS	$(4, 6, 17)$	157	73	6	3	323
4-SLOW DF-II SOS	$(4, 6, 17)$	153	105	6	4	374
3-SLOW DF-I FOS	$(2, 3, 8)$	96	43	3	1	374
4-SLOW DF-II FOS	$(2, 3, 8)$	112	43	3	2	323

FOS) can be realized using various structures, as discussed in Sections 2-4. Due to the accumulation of round-off errors, more bits are usually allocated to the internal signals of the filter than are used at the input. Our parameterizable designs allow the number of sections and also the fixed-point representation of the filter coefficients, scaling factors, and internal signals, of a chosen IIR filter structure to be adjusted based on the results of the approximation step to realize bit-true implementation of IIR filters in hardware. Because the rounding scheme can greatly influence error propagation, the implemented arithmetic blocks support various rounding modes, such as rounding to $\pm\infty$, rounding to 0, and rounding to nearest even and odd, as well as detecting the occurrence of overflows. Instantiation of the scaling multipliers can also be controlled in the HDL description of the filters.

Table 5 gives the characteristics and the implementation results of different cascade architectures for various filter structures using $M = 4$ second-order sections and in $(5, 11)$ fixed-point format. For the 3-Slow and 4-Slow designs, note that the sampling frequency is different than the operating frequency (otherwise known as the clock frequency). The clock frequency indicates the speed at which the design's internal storage elements changed their states, effectively indicating the maximum speed at which the given circuit can operate. The sampling frequency indicates how often the inputs are sampled by the design. Some applications, such as real-time signal processing, require the computation to be performed as samples appear at the input (at the sampling frequency). In such cases, the clock frequency may match the sampling frequency. However, for cases when multiple data streams are to be processed "at once", the clock frequency is usually a multiple of the sampling frequency. For example, the sampling frequency of the 3-slow design is 103 MHz and hence, it would normally have a throughput of 103 MSps if it were processing a single data stream. However, because three independent data streams can be processed at a time, the throughput is approximately three times that of a single data stream. Thus, the clock frequency is three times the sampling frequency. The results verify that the optimization transformations can increase the throughput up to six times compared to the basic IIR filter implementations.

Table 5 Characteristics and the implementation results of an eight-order IIR filter using various transformed structures on a Xilinx Virtex-7 FPGA

Design	Adders	Mults.	Delay Elements	Slice LUTs.	DSP48	Slice Regs.	Latency (Cycles)	Sampling freq. (MHz)	Throughput (MSps)
Basic DF-I	16	21	16	301	21	64	0	54	54
Basic DF-II	16	21	8	299	21	64	0	56	56
Retimed DF-I	16	21	25	301	21	256	$3+2(M-1)=9$	213	213
Retimed DF-II	16	21	25	300	21	336	$2+1(M-1)=5$	203	203
LAT DF-I	24	29	57	446	29	832	$3+2(M-1)=9$	305	305
LAT DF-II	24	29	61	446	29	896	$4+3(M-1)=13$	315	315
SLAT DF-I	32	37	81	594	37	1152	$3+2(M-1)=9$	305	305
SLAT DF-II	32	37	85	594	37	1232	$4+3(M-1)=13$	314	314
3-SLOW DF-I	16	21	65	300	21	714	$3+2(M-1)=9$	103	311
4-SLOW DF-II	16	21	65	300	21	896	$4+3(M-1)=13$	80	320

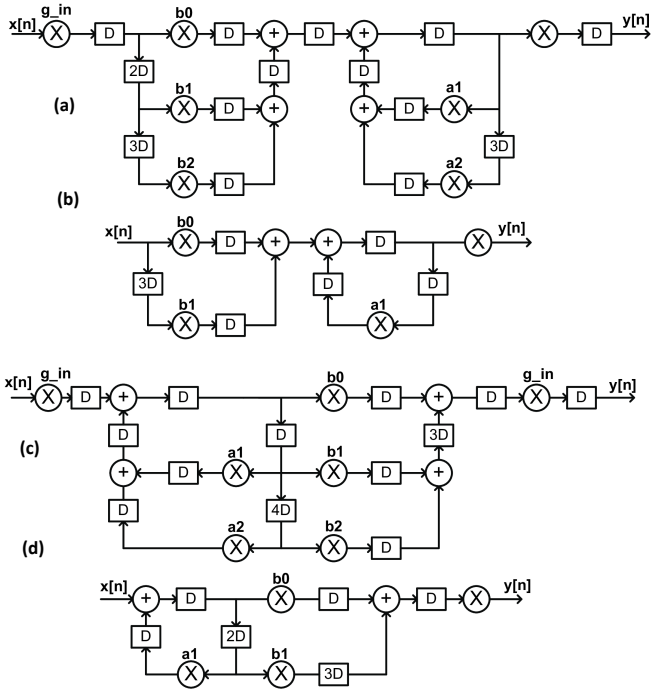

Fig. 7: (a) Retimed 3-slow SOS and (b) FOS structures in DF-I form. (c) Retimed 4-slow SOS and (d) FOS structures in DF-II form.

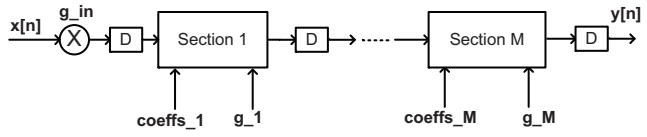

Fig. 8: Cascade IIR filter structure.

6 Compact IIR Filter Implementation using Folding Transformation

The cascade realization of an IIR filter requires M sections to achieve a sample rate equal to the design's clock frequency *CLK*. For applications where area is a primary constraint and the input data remains valid for two or more clock cycles (i.e., the sample rate is less than *CLK*), the datapath of $K < M$ sections can be reused to implement M-stage IIR filters. The folding transformation [9] can be utilized to systematically find a time-multiplexed architecture such that multiple operations of a dataflow graph are efficiently mapped onto fewer shared computational resources and hence, trading area for throughput. The mapping of the filter operations onto the folded architecture also requires a scheduler. Thus, the folded processor architecture is implemented using a time-multiplexed datapath and a control unit that schedules the operations of the filter to be executed by the shared computational units. The control unit is based on a finite state machine that periodically generates a sequence of control signals.

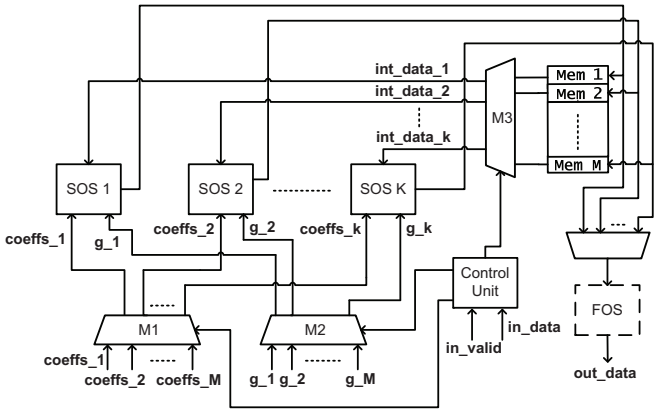

Fig. 9: The compact architecture of the reconfigurable IIR filter processor.

Fig. 9 shows the architecture of the compact reconfigurable IIR filter processor using K time-multiplexed SOSs and one FOS (for odd-ordered filters) to realize an M-stage cascade IIR filter structure. The process of calculating the filter outputs is performed iteratively. At initialization, the coefficients and scaling factors of all filter sections are loaded into a set of block memories or lookup tables (LUTs). To preserve the architectural states of the filter sections (i.e., the contents of a sections' storage elements) over different iterations, the value of each section's registers should also be stored in the block memories. The control unit selects the appropriate set of sections' coefficients, scaling factors, and sections' state variables for the corresponding K SOSs under execution at a given clock cycle (iteration). These selected values will be loaded into the corresponding sections' datapaths using three multiplexers, *M1*, *M2*, and *M3*. In the first iteration, the first K sets of filter coefficients, scalar coefficients, and intermediate data values are loaded into the K SOSs' datapaths registers. At the end of the first iteration, the intermediate data values and the output are stored in the corresponding memory blocks to be used in the next computation cycles. This sequence of operations is repeated I times until the outputs of the M-stage filter are computed. At the end of the last iteration, the output of the last section is the filtered output data $y[n]$. Hence, every time new input data arrives, the computation of the filter operations begins, and after $I = \lceil M/K \rceil$ clock cycles one output of the M-stage filter is produced. The input sample rate is determined by the number of iterations required to calculate one output value, which is $CLK/(I+1)$, where *CLK* is determined by the type of structure used in the filter's sections. For example, for the DF-I structure $CLK = 1/(T_m + 3T_a)$, for DF-II structure $CLK = 1/(2T_m + 3T_a)$, and for the retimed DF-I and DF-II SOS structures, $CLK = 1/(T_m + 2T_a)$.

Note that when implementing odd-ordered filters using a time-multiplexed SOS structure followed by a FOS, the FOS should be designed to perform data computation every I-th clock cycle. This is

Table 6 Characteristics and the implementation results of time-multiplexed filter architecture using various structures for $N = 8$ on a Xilinx Virtex-7 FPGA

Design	No. SOS	Adders	Mults.	Delay Elements	Slice Regs.	Slice LUTs.	DSP48	Latency (Cycles)	Sampling freq. (MHz)	Throughput (MSps)
Basic DF-I	1	13	7	20	325	652	6	$(M+1)(I+1)=25$	180	36
	2	24	13	21	420	1012	11	$M(I+1)=12$	180	60
	3	36	21	21	500	1599	16	$M(I+1)=12$	180	60
Basic DF-II	1	12	7	18	277	583	6	$(M+1)(I+1)=25$	120	24
	2	24	13	17	340	893	11	$M(I+1)=12$	120	40
	3	36	21	17	356	1340	16	$M(I+1)=12$	120	40
Retimed DF-I	1	13	7	22	405	744	6	$(2M+1)(I+1)=45$	211	42
	2	24	13	23	516	1110	11	$2M(I+1)=24$	211	70
	3	36	21	23	613	1768	16	$2M(I+1)=24$	211	70
Retimed DF-II	1	12	7	22	469	744	6	$(M+1)(I+1)=25$	190	37
	2	24	13	21	532	1067	11	$M(I+1)=12$	190	63
	3	36	21	21	612	1753	16	$M(I+1)=12$	190	63
LAT DF-I	1	17	9	42	1141	1587	8	$2M(I+1)=40$	350	70
	2	32	17	42	1398	2153	15	$2M(I+1)=24$	269	89
	3	48	27	42	1621	3651	22	$2M(I+1)=24$	211	70
LAT DF-II	1	17	9	42	1205	1635	8	$3M(I+1)=60$	350	70
	2	32	17	42	1462	2223	15	$3M(I+1)=36$	269	89
	3	48	27	42	1653	3720	22	$3M(I+1)=36$	210	70
SLAT DF-I	1	21	11	54	1589	2132	10	$2M(I+1)=40$	350	70
	2	40	21	54	1909	2852	19	$2M(I+1)=24$	350	117
	3	60	33	54	2197	4878	28	$2M(I+1)=24$	207	69
SLAT DF-II	1	21	10	54	1653	2180	10	$3M(I+1)=60$	350	70
	2	40	21	54	1973	2922	19	$3M(I+1)=36$	351	117
	3	60	33	54	2229	4948	28	$3M(I+1)=36$	207	69

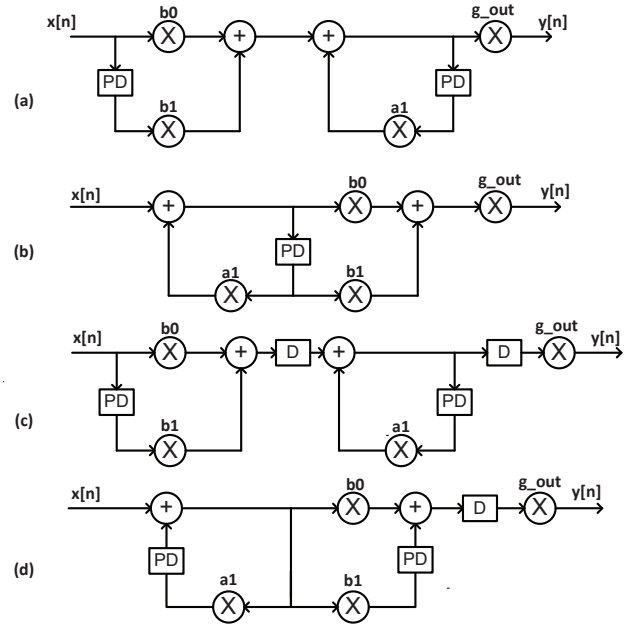

Fig. 10: (a) DF-I FOS; (b) DF-II FOS; (c) pipelined DF-I FOS; and (d) pipelined DF-II FOS.

achieved by replacing each storage element D in the FOS structures by P delay elements, where $P = I + 1$. Figs. 10(a) and (b) show the basic FOS in DF-I and DF-II structures to be cascaded with the time-multiplexed structure to implement odd-ordered filters. Basic FOSs in DF-I and DF-II structures are pipelined to achieve better performance, as shown in Figs. 10(c) and (d).

Table 6 gives the characteristics and the implementation results of the time-multiplexed architectures using different types of filter structures for an eighth-order IIR filter. Filters are implemented using $(5, 11)$ fixed-point representation, where 5 and 11 denote the integer and the fractional bit-widths of the fixed-point representation of signals, respectively. The fixed-point format of the signals in the Matlab models of the IIR filters and their Verilog descriptions are parameterizable and can be adjusted to different values. The synthesizable Verilog descriptions of the folded compact architecture were verified against their fixed-point Matlab models. Note that in the high-throughput implementation, M stages of SOSs are cascaded and the output data is computed every clock cycle. Hence, the latency of the compact design is increased compared to the direct cascade structure implementation of IIR filters. The compact designs use fewer computational resources, such as DSP48s, at the expense of lower throughputs compared to cascade implementations using M SOSs.

7 Conclusions

This article presented various high-throughput reconfigurable architectures for infinite impulse response (IIR) filters utilizing several optimization techniques, such as cutset retiming, pipelining, look-ahead transformations, and interleaving. The characteristic and implementation results of the transformed architectures on a Xilinx Virtex-7 field-programmable gate array (FPGA) are presented. The implementation results show that applying various transformations to the cascade filter structure can increase the filter's throughput up to six times compared to the basic realization of IIR filters. While direct implementation of cascade structures resulted in high-throughput processing, the hardware resources required for such realizations is directly related to the filter order, which may be unacceptable for compact implementations of relatively high-order filters. A compact reconfigurable filter processor architecture utilizing folding transformation and its characteristics and implementation results on a FPGA is also presented and discussed.

Acknowledgment

This work was supported by the Center for Sensorimotor Neural Engineering (CSNE), a National Science Foundation Engineering Research Center (EEC-1028725).

References

[1] Manolakis, D.G., Ingle, V.K.: 'Applied Digital Signal Processing: Theory and Practice', (Cambridge University Press, 2011)

[2] Oppenheim, A.V.: 'Discrete-time Signal Processing', (Pearson Education India, 1999)

[3] Liu, Y., Parhi, K.K.: 'Architectures for Recursive Digital Filters Using Stochastic Computing', IEEE Transactions on Signal Processing, 2016, 64, (14), pp. 3705 - 3718

[4] Ardakani, A., Leduc-Primeau, F., Gross, W. J.: 'Hardware Implementation of FIR/IIR Digital Filters Using Integral Stochastic Computation', IEEE International Conference on Acoustics, Speech and Signal Processing, Abu Dhabi, United Arab Emirates, 2016, pp. 6540 - 6544

[5] Hourani, R., Alassaly, H., Alexander, W..: 'Hardware Implementation of IIR Digital Filters for Programmable Devices', IEEE International Conference on Electronics, Circuits and Systems, Abu Dhabi, United Arab Emirates, 2013, pp. 783 - 786

[6] Basiri, M., Mahammad, N.: 'Configurable Folded IIR Filter Design', IEEE Transactions on Circuits and Systems - II: Express Briefs, 2015, 62, (12), pp. 1144 - 1148

[7] Leiserson, C.E., Saxe, J.B.: 'Retiming Synchronous Circuitry', Algorithmica, 1991, 6, (1), pp. 5-35

[8] Jackson, L.B.: 'Digital Filters and Signal Processing: With MATLAB Exercises', (Kluwer Academic Publishers, 2002)

[9] Parhi, K.K.: 'VLSI Digital Signal Processing Systems: Design and Implementation', (John Wiley & Sons, 1999)

[10] Parhi, K.K., Messerschmitt, D.G.: 'Pipeline Interleaving and Parallelism in Recursive Digital Filters. I: Pipelining Using Scattered Look-ahead and Decomposition', IEEE Transactions on Acoustics, Speech, and Signal Processing, 1989, 37, (7), pp. 1099 - 1117

[11] Soderstrand, M.A., de la Sema, A., Loomis, H.: 'New Approach to Clustered Look-ahead Pipelined IIR Digital Filters', IEEE Transactions on Circuits and Systems II, 1995, 42, (4), pp. 269-274

[12] Parhi, K.K., Messerschmitt, D.G.: 'Pipelined VLSI Recursive Filter Architectures Using Scattered Look-ahead and Decomposition', IEEE International Conference on Acoustics, Speech, and Signal Processing, New York, USA, 1988, pp. 2120 - 2130

[13] Khan, S.A.: 'Digital Design of Signal Processing Systems: A Practical Approach', (John Wiley & Sons, New Jersey, 2011)